

```

1
2          title 'COPYDISK 1 Feb 82'
3
4      0014          ver      equ      20      ; Version 2.0          -jrp
5
6          ; COPYDISK duplicates entire diskettes using all of the
7          ; available storage as a multiple track buffer.
8
9          ; This program must be built with a large extra segment
10         ; as follows:
11
12         ;          ASM86 COPYDISK
13         ;          GENCMD COPYDISK EXTRACMD200,XFF001
14
15         ; This allows COPYDISK to utilize all the left-over memory in your
16         ; system as its buffer.
17
18
19
20         cseg      ; code segment
21
22         start:
23         0000 BE0001      mov      si,offset signon_message
24         0003 E8FB02      0301      call    pmsg          ; print signon message
25         next_copy:
26                                     ; reload local stack for retries.
27         0006 9C          pushf          ; save interrupt flag in stack
28         0007 5B          pop      bx      ; put it in bx
29         0008 FA          cli          ; disable interrupts
30         0009 8CD8        mov      ax,ds      ; get our data segment
31         000B 3ED0        mov      ss,ax      ; and use as stack segment
32         000D BCD204       mov      sp,offset stack_end ; set stack pointer
33         0010 53          push     bx      ; flags back into stack
34         0011 9D          popf         ; restore interrupt status
35
36         0012 BE3501      mov      si,offset source_message ; prompt for source drive
37         0015 E8A802      02C0      call    get_drive      ; request drive code
38         0018 3CCC        cmp      al,cr - 'A' ; see if it was a <cr>
39         001A 7503      001F      jne      next_1          ; no, continue
40         001C E95001      016F      jmp      exit          ; yes, we should exit
41         next_1:
42         001F 3C10        cmp      al,drive_cnt ; see if valid drive code
43         0021 7209      002C      jb      save_source      ; yes, go save it
44         drive_err:
45         0023 BE7F02      mov      si,offset bad_drive ; else, print
46         0026 E8D802      0301      call    pmsg          ; a bad drive message
47         0029 E93801      0164      jmp      another      ; and try to get another
48
49         save_source:
50         002C A25103      mov      source,al ; save it as the source drive
51         002F 0441      add      al,'A' ; make ascii drive code
52         0031 A29001      mov      source_ascii,al ; save for messages
53         0034 8A0E5103    mov      cl,source ; get source drive

```

CP/M-86 v.1.1 (Vol. II)

see if valid drive code
yes, go save it: 1981, 1982

else, print RESEARCH

a bad drive message
and try to get another

PACIFIC GROVE, CA 93953

C81-162
SER: save it as the source drive

make ascii drive code
save for messages

get source drive

```

54
55 0038 B200          mov     dl,0                ; force first time selection
56 003A E85002      028D    call    seldsk          ; select it
57 003D 85DB        test     bx,bx                ; insure the drive exists
58 003F 74E2      0023    jz      drive_err          ; no, fatal
59 0041 268B770A    mov     si,ES:word ptr 10[bx]    ; get pointer to DPB
60                      ; create local copy of disk definition
61 0045 1E06        push ds ! push es              ; save segments
62 0047 1F07        pop ds ! pop es                ; and exchange them
63 0049 BFC303      mov     di,offset disk_def      ; point to local disk def table
64 004C B90F00      mov     cx,15                  ; diskdef is 15 bytes long
65 004F FCF3A4      cld ! rep movsb                ; copy it
66 0052 1E06        push ds ! push es              ; swap back
67 0054 1F07        pop ds ! pop es                ; segment registers
68
69 0056 268B17      mov     dx,es:word ptr 00[bx]    ; get secttran table address
70 0059 B90000      mov     cx,0                    ; logical sector zero
71 005C EB4A02      02A9    call    sectran          ; find out first sector number
72 005F 891E3803    mov     base_sector,bx          ; save this for track reads
73
74
75 0063 BE5A01      mov     si,offset dest_message  ; prompt for destination
76 0066 E85702      02C0    call    get_drive        ; get the drive code
77 0069 3A065103    cap     al,source                ; see if same as source
78 006D 7509      0078    jne     not_same            ; no, go see if > max
79 006F BE9A02      mov     si,offset same_message   ; can't have same drive
80 0072 E83C02      0301    call    msgq              ; so we print message
81 0075 E9EC00      0164    jmp     another            ; and go try again
82
83                      not_same:
84 0078 3C10      cap     al,drive_cnt                ; is dest > max?
85 007A 73A7      0023    jnb     drive_err          ; go print invalid drive error
86
87 007C A25203      mov     dest,al                    ; save destination drive
88 007F 0441      add     al,'A'                  ; make ascii drive code
89 0081 A29B01      mov     dest_ascii,al            ; save for messages
90
91 0084 8A0E5203    mov     cl,dest                ; get destination drive
92 0088 B200      mov     dl,0                    ; force first time selection
93 008A EB0002      028D    call    seldsk          ; select destination drive
94 008D 85DB        test     bx,bx                ; insure drive exists
95 008F 7492      0023    jz      drive_err          ; no, go error out
96 0091 268B7F0A    mov     di,ES:word ptr 10[bx]    ; point to destination DPB
97 0095 BEC303      mov     si,offset disk_def      ; point to source drives def
98 0098 B90F00      mov     cx,15                  ; length of a definition
99 009B FCF3A6      cld ! rep cmpsb                ; compare the definitions
100 009E 7403      00A3    je      next_2                ; must be equal.
101 00A0 E9DC00      017F    jmp     different_type          ; else go print error
102                      next_2:
103 00A3 268B17      mov     dx,es:word ptr 00[bx]    ; get secttran table address
104 00A6 B90000      mov     cx,0                    ; logical sector zero
105 00A9 EBFD01      02A9    call    sectran          ; find out first sector number
106 00AC 3B1E6303    cap     bx,base_sector            ; make sure its the same

```

; prompt for destination

; get the drive code

; see if same as source

; no, go see if > max

; can't have same drive

; so we print message

; and go try again

.SER. =

; is dest > max?

; go print invalid drive error

; save destination drive

; make ascii drive code

; save for messages

; get destination drive

; force first time selection

; select destination drive

; insure drive exists

; no, go error out

; point to destination DPB

; point to source drives def

; length of a definition

; compare the definitions

; must be equal.

; else go print error

; get secttran table address

; logical sector zero

; find out first sector number

; make sure its the same

```

107
108 00B0 7403      00B5      je      next_3          ; the same is OK,
109 00B2 E7CA00    017F      jmp      different_type ; no, fatal error
110
111                      next_3:                      ; determine track capacity
112 00B5 A1C303      mov      ax,spt                    ; get the sectors per track
113 00BB B2B0        mov      dl,128                  ; length of a sector
114 00BA F6E2        mul      dl                      ; get track capacity
115 00BC A35C03      mov      track_size,ax           ; save it
116
117                      ; determine number of tracks on diskette
118 00BF A0C603      mov      al,blm                  ; get block mask
119 00C2 FFC0        inc      al                      ; plus 1 gives sectors/block
120 00C4 B400        mov      ah,0                    ; make 16 bits
121 00C6 BA0000      mov      dx,0                    ; make 32 bits
122 00C9 F726C303    mul      dsa                    ; total sectors/data area
123 00CD 0306C303    add      ax,spt                    ; force round up
124 00D1 4B          dec      ax                      ; by adding SPT-1
125 00D2 F736C303    div      spt                    ; compute number of tracks
126 00D6 0306D003    add      ax,off                  ; add in operating sys tracks
127 00DA 4B          dec      ax                      ; number of last track
128 00DB A36A03      mov      max_track,ax           ; save value
129
130                      ; compute number of tracks that will
131                      ; fit in our data segment
132 00DE A10C00      mov      ax,extra_length          ; get low 16 bits of DS length
133 00E1 8A160E00    mov      dl,extra_length_H        ; get other 8 bits
134 00E5 B600        mov      dh,0                    ; zero rest of 32 bit divisor
135 00E7 F7365C03    div      track_size              ; divide buff_len / track_size
136 00EB A35803      mov      NTS,ax                  ; save this value.
137 00EE 3D0200      cmp      ax,2                    ; must be at least 2 tracks
138 00F1 730F      0102      jae      adequate_memory ; ok, continue
139 00F3 BEC401      mov      si,offset memory_message ; else print
140 00F6 EB0802      0301      call     pmsg           ; error message
141
142                      aborting:                      ; here if exiting because of error
143 00F9 BEED01      mov      si,offset abort_message
144 00FC EB0202      0301      call     pmsg
145 00FF E96200      0164      jmp      another
146
147                      adequate_memory:
148
149
150 0102 B67F01      mov      si,offset ready_message ; insure this is correct
151 0105 E8CF01      02D7      call     get_yn         ; print request
152 0108 72EF      00F9      jc      aborting         ; if NO, then get new input
153 010A BE3102      mov      si,offset copy_message ; else tell user OK.
154 010D EBF101      0301      call     pmsg
155
156 0110 A16A03      mov      ax,max_track            ; get maximum track number
157 0113 A35A03      mov      base_track,ax           ; save as outer loop index
158                      next_block:
159 0116 A15A03      mov      ax,base_track            ; get loop index

```

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH

0007-379
0005-111-00950
SEEK

```

160
161 0119 3D0000      cmp     ax,0           ; see if we are outside limits
162 011C 7C40      015E    jl     next_block_x      ; yes, terminate loop
163 011E 2B065803   sub     ax,nts         ; lower by number of tracks per buffer
164 0122 40        inc     ax             ; and adjust by 1.
165 0123 7903      0128    jns     not_neg         ; check to make sure its positive
166 0125 B30000     mov     ax,0           ; no, zero is floor
167               not_neg:
168 012B A35E03     mov     btr,ax          ; save as last track for this copy page
169
170 012B BE4202     mov     si,offset reading_message ; point to read message
171 012E B3A102     mov     ax,offset read      ; and proper subroutine
172 0131 8A0E5103   mov     cl,source          ; select source diskette
173 0135 E85000      0188    call    track_block
174
175 0138 BE5402     mov     si,offset writing_message ; point to write message
176 013B B3A502     mov     ax,offset write      ; and proper subroutine
177 013E 8A0E5203   mov     cl,dest             ; select destination
178 0142 E84300      0188    call    track_block
179
180 0145 BE6602     mov     si,offset verify_message ; point to verify message
181 0148 B36302     mov     ax,offset verify
182 014B 8A0E5203   mov     cl,dest             ; reselect destination
183 014F E83600      0188    call    track_block
184
185 0152 A15A03     mov     ax,base_track
186 0155 2B065803   sub     ax,nts                 ; adjust base track by number we copied
187 0159 A35A03     mov     base_track,ax
188 015C EB88      0116    jmps    next_block
189
190               next_block_x:           ; here, we are done with a successful copy
191 015E BE0002     mov     si,offset done_message ; print message
192 0161 EB9D01      0301    call    pmsg          ; announcing success
193
194               another:
195 0164 BE1202     mov     si,offset another_message
196 0167 EB6D01      02D7    call    getyn          ; see if we want another copy
197 016A 7203      016F    jc     exit             ; no, go exit
198               next_copy_v:
199 016C E997FE      0006    jmp     next_copy        ; convenient for short jumps
200               ; back for another disk copy
201
202               exit:
203 016F BE3803     mov     si,offset exit_message ; here, we are exiting
204 0172 EB8C01      0301    call    pmsg
205
206 0175 B10D     mov     cl,13
207 0177 CDE0     int     BDOS
208
209 0179 B100     mov     cl,0
210 017B B200     mov     dl,0
211 017D CDE0     int     BDOS
212

```

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH

reboot P. O. BOX 579
PACHA GROVE, CA 95350

SER. #

```

213
214                different_type:
215 017F BEC802      mov     si,offset type_error_message
216 0182 EB7C01      0301    call    pmsg
217 0185 E971FF      00F9    jmp     aborting
218
219                track_block:
220 0188 89366403     mov     message_pointer,si      ; save pointer to message
221 018C A36603       mov     disk_function,ax        ; save pointer to read/write
222 018F 880E5303     mov     selected_disk,cl        ; save drive select code
223 0193 B201         mov     dl,1                    ; tell bios have selected before
224 0195 E8F500      028D    call    seldsk           ; select diskette drive
225 0198 A15A03       mov     ax,base_track          ; get base track number for this pass
226 019B A35403       mov     trk,ax                 ; save as loop index
227                                     ; first inner loop...
228 019E A15403       mov     ax,trk                 ; get the index
229 01A1 JB065E03     cmp     ax,btr                 ; see if less than last
230 01A5 7D03        01AA    jnl     next_4           ; no, continue
231 01A7 E9B800      0262    jmp     next_track_x    ; yeah, exit loop
232                next_4:
233 01AA 3B366403     mov     si,message_pointer
234 01AE E85001      0301    call    pmsg
235 01B1 A15403       mov     ax,trk
236 01B4 EB6A01      0321    call    pdec
237 01B7 BE7802       mov     si,offset space_message
238 01BA EB4401      0301    call    pmsg
239 01BD 880E5403     mov     cx,trk
240 01C1 EBCD00      0291    call    settrk
241 01C4 B90000       mov     cx,0
242                next_sector:
243 01C7 51           push    cx                    ; save current sector number
244 01CB 890E5603     mov     sector,cx                 ; save sector number (0 org)
245 01CC 030E6803     add     cx,base_sector          ; correct for first sector number
246 01D0 EBC200      0295    call    setsec          ; pass to cbios
247
248                                     ; compute dma address and segment
249 01D3 A15A03       mov     ax,base_track          ; get starting track number
250 01D6 2B065403     sub     ax,trk                 ; gives relative track of pass
251 01DA B80000       mov     dx,0                    ; make dword
252 01DD F726C303     mul     spt                    ; compute sector of track
253 01E1 03065603     add     ax,sector                ; add in current logical sector
254 01E5 83D200       adc     dx,0                    ; make double precision add
255 01E8 878000       mov     cx,128                ; length of a sector
256 01EB F7E1        mul     cx                    ; gives base offset as 32 bits
257 01ED A36003       mov     dma_offset,ax          ; gives base dma offset
258 01F0 B10CD3E2     mov     cl,12 ; shl dx,cl        ; move to high nibble
259 01F4 03160F00     add     dx,extra_base          ; offset extra segment
260 01F8 89166203     mov     dma_segment,dx        ; save DMA segment
261 01FC 3BCA        mov     cx,dx                    ; put in argument register
262 01FE EB9C00      029D    call    setdmab         ; pass to CBIOS
263 0201 3B0E6003     mov     cx,dma_offset          ; get the offset again
264 0205 EB9100      0299    call    setdma          ; and pass it to BIOS also
265

```

```

; print <xxx> track
; get the track number
; print as decimal
; then, print some
; spaces to blank any garbage
; get desired track address
; give to bios
; start with logical sector 0

```

```

266
267 020B A15603      mov    ax,sector      ; fetch current sector number
268 020B 40          inc     ax             ; add one to it
269 020C 3B06C303    cmp     ax,spt        ; see if last sector on track
270 0210 B100        mov     cl,0          ; might be then normal write
271 0212 7202        jnb     execute_function ; skip if not last
272 0214 8101        mov     cl,1          ; if last, then treating as
273                                     ; dir write forces flush
274
275 0216 FF166603    execute_function:      call    disk_function      ; read/write/verify a sector
276 021A 84C0        test    al,al         ; check error return code
277 021C 7432        jz      no_disk_error ; see if we got a bad sector
278
279                                     ; got fatal disk error
280 021E BE2203      mov     si,offset disk_error_message
281 0221 E8D000      call    pmsg           ; print disk error
282 0224 8B366403    mov     si,message_pointer ; get pointer to read/write
283 0228 46          inc     si             ; skip leading <cr>
284 0229 E8D500      call    pmsg           ; print that
285 022C A15403      mov     ax,trl         ; get track number
286 022F E8EF00      call    pdec          ; print it
287 0232 B11803      mov     si,offset sector_message ; print ", Sector "
288 0235 E8C900      call    pmsg
289 0238 A15603      mov     ax,sector      ; get sector number (0 org)
290 023B 03066803    add     ax,base_sector ; correct if 1 origin
291 023F E8DF00      call    pdec          ; print the sector number
292 0242 BE0003      mov     si,offset continue_message ; see if user wants to ignore
293 0245 E88F00      call    get_yn         ; ask for a Y/N response
294 0248 7303        jnc     next_5         ; yes, continue
295 024A E9ACFE      jmp     aborting      ; NO, go ask for new disks
296
297 024D E8AE00      next_5:               call    crlf
298                                     no_disk_error:
299 0250 59          pop     cx             ; recover sector number
300 0251 41          inc     cx             ; sector = sector + 1
301 0252 3B0EC303    cmp     cx,spt        ; see if past last sector
302 0256 7303        jae     next_6         ; done, exit RESEARCH
303 0258 E96CFF      jmp     next_sector ; else, continue
304                                     next_6:
305 025B FF0E5403    dec     trk             ; track = track - 1
306 025F E93CFF      jmp     next_track    ; continue loading buffer
307                                     next_track_x:
308 0262 C3          ret
309
310
311
312
313
314 0263 B9D204      verify:               ; verify a 128 byte sector
315 0266 E83000      call    setdma         ; point at our 128 byte buffer
316 0269 0CD9        mov     cx,ds         ; make it the dma buffer
317 026B E82F00      call    setdma         ; get our data segment
318 026E E83000      call    read          ; point dma base address
                                     ; read 128 byte record

```

```

; recover sector number
; sector = sector + 1
; see if past last sector
; done, exit RESEARCH
; else, continue
; track = track - 1
; continue loading buffer
SER #

```

```

319
320 0271 50          push    ax          ; save read error in stack
321 0272 BED204      mov     si,offset verify_buffer ; point to sector we just read
322 0275 C43E6003    les     di,dword ptr dma_offset ; get pointer to one we wrote
323 0279 B73000      mov     cx,128      ; 128 byte record
324 027C FCF3A6      cld ! rep cmpsb    ; compare them
325 027F B000        mov     al,0        ; no error
326 0281 7402        je       verify_good ; so we are ok
327 0283 B0FF        mov     al,0FFh     ; else, we got compare error
328                verify_good:
329 0285 5A          pop     dx          ; recover read error byte
330 0286 0AC2        or      al,dl      ; merge into return code
331 0288 C3          ret              ; back to caller
332
333
334                ; *****
335                ;
336                ;      BIOS direct entry points
337                ;
338                ; *****
339
340                home:
341 0289 B008          mov     al,8
342 028B EB20          02AD     jmps   bios_call
343
344                seldsk:
345 028D B009          mov     al,9
346 028F EB1C          02AD     jmps   bios_call
347
348                settrk:
349 0291 B00A          mov     al,10
350 0293 EB18          02AD     jmps   bios_call
351
352                setsec:
353 0295 B00B          mov     al,11
354 0297 EB14          02AD     jmps   bios_call
355
356                setdma:
357 0299 B00C          mov     al,12
358 029B EB10          02AD     jmps   bios_call
359
360                setdmab:
361 029D B011          mov     al,17
362 029F EB0C          02AD     jmps   bios_call
363
364                read:
365 02A1 B00D          mov     al,13
366 02A3 EB08          02AD     jmps   bios_call
367
368                write:
369 02A5 B00E          mov     al,14
370 02A7 EB04          02AD     jmps   bios_call
371

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

372
373          sectran:
374 02A9 B010          mov     al,16
375 02AB EB00          02AD    jmps   bios_call
376
377          bios_call:
378 02AD A2BE03          mov     bios_function,al
379 02B0 890EBF03        mov     bios_CX,cx
380 02B4 8916C103        mov     bios_DX,dx
381 02B8 F132            mov     cl,50
382 02BA BADE03          mov     dx,offset bios_function
383 02BD CDE0            int     BDOS
384 02BF C3              ret
385
386
387          ; *****
388          ; *
389          ; *      Operator interaction subroutines
390          ; *
391          ; *****
392
393          get_drive:          ; print message and read drive code
394 02C0 E83E00          0301    call    pmsg          ; first, print the message
395 02C3 E80300          02C9    call    get_upper      ; then get an upper case letter
396 02C6 2C41            sub     al,'A'          ; and normalize to 0...
397 02C8 C3              ret
398
399          get_upper:          ; get upper case console input w/ echo
400 02C9 E81D00          02E9    call    get_char          ; get console character
401 02CC 3C61            cmp     al,'a'          ; see if lower case
402 02CE 7206            02D6    jb     get_upper_x      ; below, leave
403 02D0 3C7A            cmp     al,'z'          ; insure not > z
404 02D2 7702            02D6    ja     get_upper_x      ; not lower at all
405 02D4 2C20            sub     al,'a'-'A'        ; make upper
406
407          get_upper_x:
408          ret
409
410          get_yn:              ; print message and get Y or N
411 02D7 56              push    si          ; save message pointer
412 02DB E82600          0301    call    pmsg          ; print the message
413 02DB E8EBFF          02C9    call    get_upper      ; get a upper case letter
414 02DE 5E              pop     si          ; recover message pointer
415 02DF 3C59            cmp     al,'Y'          ; see if response was 'Y'
416 02E1 7405            02E8    je     get_yn_x          ; yes, return w/ no carry
417 02E3 3C4E            cmp     al,'N'          ; see if was 'N'
418 02E5 75F0            02D7    jne    get_yn          ; no, invalid. reprompt
419 02E7 F9              stc          ; set carry for a NO
420          get_yn_x:
421          ret
422
423          get_char:              ; read a line from CONIN and return first char
424 02E9 D10A            mov     cl,10          ; function for line in
425 02EB BA6C03          mov     dx,offset line_buff      ; point at buffer

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____


```

425
426 02EE CDE0          int      BDOS                ; read the line
427 02F0 A06D03        mov      al,line_buff+1      ; get input length
428 02F3 3C01          cmp      al,1                ; insure single char entered
429 02F5 7204          jb       get_char_1           ; no, go
430 02F7 A06E03        mov      al,line_buff+2      ; get the character
431 02FA C3            ret
432                  get_char_1:
433 02FB B00D          mov      al,cr                ; null string, return a <CR>
434 02FD C3            ret
435
436
437                  crlf:                          ; print a <CR><LF>
438 02FE BE3503        mov      si,offset crlf_message
439                                     ; fall into pmsg
440
441                  pmsg:                          ; print message at [SI] to zero
442 0301 AC            lods      al                    ; get a character
443 0302 84C0          test     al,al                  ; see if zero terminator
444 0304 7408          jz       pmsg_x                ; yes, exit
445 0306 56            push     si                    ; save pointer
446 0307 E81000        call     conout                 ; not done, print character
447 030A 5E            pop      si                    ; recover pointer
448 030B E9F3FF        jmp      pmsg                 ; and loop
449                  pmsg_x:
450 030E C3            ret
451
452                  conin:                          ; get character from console into AL
453 030F B106          mov      cl,6
454 0311 B2FF          mov      dl,0FFh
455 0313 CDE0          int      BDOS
456 0315 84C0          test     al,al
457 0317 74F6          jz       conin
458 0319 C3            ret
459
460                  conout:
461 031A 8AD0          mov      dl,al
462 031C B106          mov      cl,6
463 031E CDE0          int      BDOS
464 0320 C3            ret
465
466                  pdec:                          ; print unsigned 16 bits in AX as decimal
467                                     ; with zero suppression
468 0321 B90000        mov      cx,0                  ; cx is digit counter
469                  pdec_1:                        ; here to divide out next digit
470 0324 2BD2          sub      dx,dx                  ; Zero DX
471 0326 8B0A00        mov      bx,10                 ; constant 10
472 0329 F7F3          div      bx                      ; quotient to AX, remainder to DX
473 032B 80C230        add      dl,'0'                ; make remainder ascii digit
474 032E 52            push     dx                    ; and stick onto stack
475 032F 41            inc      cx                    ; bump digit counter
476 0330 85C0          test     ax,ax                  ; see if any quotient left
477 0332 75F0          jnz      pdec_1                ; yes, continue stacking digits

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

; output character in AL to console

PACIFIC GROVE, CA 95660

SER. #

```

473
479          pdec_2:
480 0334 5B          pop     ax          ; get a digit from the stack
481 0335 51          push    cx         ; save count
482 0336 E8E1FF      031A      call   conout      ; print it
483 0339 59          pop     cx         ; restore count
484 033A E2F8      0334      loop   pdec_2       ; and continue if more in stack
485 033C C3          ret              ; done. . .
486
487
488
489          ; ** DATA SEGMENT **
490          dseg
491
492 000D          cr      equ     0dh        ; ascii carriage return
493 000A          lf      equ     0ah        ; ascii line feed
494
495 0010          drive_cnt equ 16          ; CP/M currently supports up to 16 drives
496
497 00E0          BDOS    equ     224        ; system call interrupt number
498
499
500          ; CP/M-86 Page Zero
501
502 0000          code_length rw 1          ; low 16 bits code length
503 0002          code_length_H rb 1        ; high 8 bits " "
504 0003          code_base rw 1          ; base of the code segment
505 0005          model_8080 rb 1          ; 8080 memory model flag
506 0006          data_length rw 1         ; low 16 bits data length
507 0008          data_length_H rb 1        ; high 8 bits " "
508 0009          data_base rw 1          ; base of the data segment
509 000B          rs 1                    ; not used
510 000C          extra_length rw 1         ; low 16 bits extra length
511 000E          extra_length_H rb 1       ; high 8 bits " "
512 000F          extra_base rw 1          ; base of the extra segment
513
514
515          org     005Ch
516
517 005C          default_FCB rs 35        ; default File Control Block
518
519          org     0080h
520
521 0080          default_buffer rs 128     ; default record buffer
522
523          org     0100h
524
525          ; start of user data segment
526
527          ; MESSAGES
528 0100 000A43502F4D signon_message db cr,lf,'CP/M-86 Full Disk COPY Utility'
529      2D3836204675
530      6C6C20446973

```

```

531
532      6B20434F5059
533      205574696C39
534      7479
535 0120 0D0A20202020      db      cr,lf,'      Version '
536      205665727369
537      6F6E20
538 012F 322E300D0A00      db      ver/10+'0', '.', (ver mod 10)+'0',cr,lf,0
539
540 0135 0D0A0D0A456E      source_message db      cr,lf,cr,lf,'Enter Source Disk Drive (A-D) ? ',0
541      74657220536F
542      757263652044
543      69736B204472
544      697665202841
545      2D4429203F20
546      00
547
548 015A 0D0A0D0A2044      dest_message db      cr,lf,cr,lf,'Destination Disk Drive (A-D) ? ',0
549      657374696E61
550      74696F6E2044
551      69736B204472
552      697665202841
553      2D4429203F20
554      00
555
556 017F 0D0A0D0A436F      ready_message db      cr,lf,cr,lf,'Copying Disk '
557      7079696E6720
558      4469736B20
559 0190      source_ascii  rb      1
560 0191 3A20746F2044      db      ': to Disk '
561      69736B20
562 019B      dest_ascii   rb      1
563 019C 3A0D0A      db      ':',cr,lf
564 019F 497320746869      db      'Is this what you want to do (Y/N) ? ',0
565      732077686174
566      20796F752077
567      616E7420746F
568      20646F202859
569      2F4E29203F20
570      00
571
572 01C4 0D0A496E7375      memory_message db      cr,lf,'Insufficient memory available for copy',0
573      666669636965
574      6E74206D656D
575      6F7279206176
576      61696C61626C
577      6520666F7220
578      636F707900
579
580 01ED 0D0A0D0A436F      abort_message db      cr,lf,cr,lf,'Copy aborted',cr,lf,0
581      70792061626F
582      727465640D0A
583      00

```

COPYDISK 1.1 1982

PACIFIC GROVE, CA 93950

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. #

```

584
585
586 0200 0D0A436F7079 done_message db cr,lf,'Copy completed.',0
587 20636F6D706C
588 657465642E00
589
590 0212 0D0A0D0A436F another_message db cr,lf,cr,lf,'Copy another disk (Y/N) ? ',0
591 707920616E6F
592 746865722064
593 69736B202859
594 2F4E29203F20
595 00
596
597 0231 0D0A436F7079 copy_message db cr,lf,'Copy started',cr,lf,0
598 207374617274
599 65640D0A00
600
601 0242 0D2020526561 reading_message db cr,' Reading Track ',0
602 64696E6372054
603 7261636B2000
604
605 0254 0D2020577269 writing_message db cr,' Writing Track ',0
606 74696E6372054
607 7261636B2000
608
609 0266 0D5665726966 verify_message db cr,'Verifying Track ',0
610 79696E6372054
611 7261636B2000
612
613 0278 202020202020 space_message db ' ',0
614 00
615
616 027F 0D0A496C6C65 bad_drive db cr,lf,'Illegal Diskette Drive',cr,lf,0
617 67616C204469
618 736B65747465
619 204472697365
620 0D0A00
621
622 029A 0D0A536F7572 same_message db cr,lf,'Source and Destination cannot be the same'
623 636520616E64
624 204465737469
625 6E6174696F6E
626 2063616E6E6F
627 742062652074
628 68652073616D
629 65
630 02C5 0D0A00 db cr,lf,0
631
632 02C8 0D0A536F7572 type_error_message db cr,lf,'Source and Destination disks must be'
633 636520616E64
634 204465737469
635 6E6174696F6E
636 206469736B73

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

637
638      206D75737420
639      6265
640 02EE 0D0A74686520          db      cr,lf,'the same type',cr,lf,0
641      73616D652074
642      7970650D0A00
643
644 0300 0D0A9676E6F          continue_message db      cr,lf,'Ignore error (Y/N) ? ',0
645      726520657272
646      6F722028592F
647      4E29203F2000
648
649 0318 2C2053656374          sector_message db      ', Sector ',0
650      6F722000
651
652 0322 0D0A5065726D          disk_error_message db      cr,lf,'Permanent Error ',0
653      616E656E7420
654      4572726F7220
655      00
656
657 0335 0D0A00          crlf_message      db      cr,lf,0
658
659 0338 0D0A434F5059          exit_message      db      cr,lf,'COPY program exiting',cr,lf,0
660      2070726F6772
661      616D20657869
662      74696E670D0A
663      00
664
665
666
667

```

; VARIABLES

```

668 0351          source      rb      1      ; source drive select code
669 0352          dest        rb      1      ; destination drive select code
670 0353          selected_disk rb      1      ; current drive select code
671 0354          trk         rw      1      ; current track number
672 0356          sector      rw      1      ; current sector number (0 origin)
673 0358          nts        rw      1      ; number of tracks per buffer full
674 035A          base_track  rw      1      ; starting track number for this pass
675 035C          track_size  rw      1      ; size of each track in bytes
676 035E          btr        rw      1      ; last track number of this pass
677 0360          dma_offset  rw      1      ; current buffer offset
678 0362          dma_segment rw      1      ; current buffer segment base
679 0364          message_pointer rw      1      ; points to appropriate read/write message
680 0366          disk_function rw      1      ; points to " " " subroutine
681 0368          base_sector  rw      1      ; either 0 or 1 normally
682 036A          max_track   rw      1      ; total number of tracks on disk
683
684 036C 50          line_buff  db      80
685 036D 00          db      0
686 036E          rb      80
687
688 038E          bios_function rb      1      ; bios function number
689 03BF          bios_cx     rw      1      ; first argument for BIOS call

```

CP/M-86 v.1.1 (Vol. II)

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # C81-162

CP/M ASM86 1.1 SOURCE: COPYDISK.A86 COPYDISK 1 Feb 82

```

690
691 03C1      bios_dx      rw      1      ; second argument for BIOS call
692
693          disk_def      rs      0      ; disk definition table gets copied here
694 03C3          spt      rw      1      ; 128 byte sectors per track
695 03C3          bsh      rb      1      ; block shift factor
696 03C5          blm      rb      1      ; block mask
697 03C6          exm      rb      1      ; extent mask
698 03C7          dsm      rw      1      ; disk size in blocks
699 03C8          drn      rw      1      ; directory size
700 03CA          alo      rb      1      ; alloc 0
701 03CC          all      rb      1      ; alloc 1
702 03CD          cks      rw      1      ; checksum size
703 03CE          off      rw      1      ; directory offset
704 03D0
705
706 03D2          stack_end  rw      0
707 04D2
708
709          verify_buffer  rs      128   ; sector buffer for verify function
710 04D2
711          db      0      ; force out data segment
712 0552 00
713
714          end
715
716
717
718 END OF ASSEMBLY.  NUMBER OF ERRORS: 0.  USE FACTOR: 10%

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

ABORTING	00F9 L	142#	152	217	295						
ABORTMESSAGE	01ED V	143	580#								
ADEQUITEMEMORY	0102 L	138	148#								
ALO	03CC V	701#									
AL1	03CD V	702#									
ANOTHER	0164 L	46	81	145	174#						
ANOTHERMESSAGE	0212 V	195	590#								
BADDRIVE	027F V	44	616#								
BASESECTOR	0368 V	72	106	245	290	681#					
BASETRACK	075A V	157	159	185	187	225	249	674#			
BDOS	00E0 N	207	211	383	426	455	463	497#			
BIOSCALL	02AD L	342	346	350	354	358	362	366	370	375	377#
BIOSCX	03BF V	379	689#								
BIOSDX	03C1 V	380	691#								
BIOSFUNCTION	03BE V	378	382	688#							
BLM	07C6 V	118	697#								
BSH	03C5 V	696#									
BTR	03SE V	168	229	676#							
CKS	03CE V	703#									
CODEBASE	0003 V	504#									
CODELENGTH	0000 V	502#									
CODELENGTHH	0002 V	503#									
CONIN	030F L	452#	457								
CONOUT	031A L	446	460#	482							
CONTINUEMESSAGE	0300 V	292	644#								
COPYMESSAGE	0231 V	153	597#								
CR	000D N	37	433	492#	528	535	538	540	540	548	548
		556	556	563	572	580	580	580	586	590	590
		597	597	601	605	609	616	616	622	630	632
		640	640	644	652	657	659	659			
CRLF	02FE L	297	437#								
CRLFMESSAGE	0335 V	438	657#								
DATABASE	0009 V	503#									
DATALNGTH	0006 V	506#									
DATALNGTHH	0008 V	507#									
DEFAULTBUFFER	0080 V	521#									
DEFAULTFCB	005C V	517#									
DEST	0352 V	87	91	177	182	669#					
DESTASCII	019B V	89	562#								
DESTMESSAGE	015A V	75	548#								
DIFFERENTTYPE	017F L	101	109	214#							
DISKDEF	03C3 V	63	97	694#							
DISKERRORMESSAGE	0322 V	280	652#								
DISKFUNCTION	0366 V	221	275	680#							
DMIOFFSET	0360 V	257	263	322	677#						
DMASEGMENT	0362 V	260	678#								
DONEMESSAGE	0200 V	191	586#								
DRIVECNT	0010 N	41	84	495#							
DRIVEERR	0023 L	43#	58	85	95						
DRM	03CA V	700#									
DSM	03C9 V	122	699#								
EXECUTEFUNCTION	0216 L	271	274#								
EXIT	016F L	39	197	202#							

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. Box 579

PACIFIC GROVE, CA 93950

SER. # _____

EXITMESSAGE	0338 V	203	659#										
EXN	03C7 V	698#											
EXTRABASE	000F V	259	512#										
EXTRALENGTH	000C V	132	510#										
EXTRALENGTHH	000E V	133	511#										
GETCHAR	02E9 L	400	422#										
GETCHAR1	02FB L	429	432#										
GETDRIVE	02C0 L	36	76	393#									
GETUPPER	02C9 L	395	399#	412									
GETUPPERX	02D6 L	402	404	406#									
GETYN	02D7 L	151	196	293	409#	417							
GETYNX	02EB L	415	419#										
H0ME	0289 L	340#											
LF	000A N	493#	528	535	538	540	540	548	548	556	556		
		563	572	580	580	580	586	590	590	597	597		
		616	616	622	630	632	640	640	644	652	657		
		659	659										
LINEBUF	036C V	424	427	430	684#								
MAXTRACK	036A V	128	156	682#									
MEMORYMESSAGE	01C4 V	139	572#										
MESSAGEPOINTER	0364 V	220	233	282	679#								
MODEL8080	0005 V	505#											
NEXT1	001F L	38	40#										
NEXT2	00A3 L	100	102#										
NEXT3	00B5 L	108	111#										
NEXT4	01AA L	230	232#										
NEXT5	024D L	294	296#										
NEXT6	025B L	302	304#										
NEXTBLOCK	0116 L	158#	188										
NEXTBLOCKX	015E L	162	190#										
NEXTCOPY	0006 L	24#	179										
NEXTCOPYV	016C L	198#											
NEXTSECTOR	01C7 L	242#	303										
NEXTTRACK	019E L	227#	306										
NEXTTRACKX	0262 L	231	307#										
NODISKERROR	0250 L	277	298#										
NOTNEG	0128 L	165	167#										
NOTSAME	0078 L	78	83#										
NTS	0358 V	136	163	186	673#								
OFF	03D0 V	126	704#										
PDEC	0321 L	236	286	291	466#								
PDEC1	0324 L	469#	477										
PDEC2	0334 L	479#	484										
PSG	0301 L	23	45	80	140	144	154	192	204	216	234		
		238	201	284	288	394	411	441#	448				
PSGX	030E L	444	447#										
READ	02A1 L	171	318	364#									
READINGMESSAGE	0242 V	170	601#										
READYMESSAGE	017F V	150	556#										
SAMEMESSAGE	029A V	79	622#										
SAVESOURCE	002C L	42	48#										
SECTOR	0356 V	244	253	267	289	672#							
SECTORMESSAGE	0318 V	287	649#										

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

SECTRA	02A9 L	71	105	373					
SELOSK	028D L	56	93	224	344				
SELECTEDDISK	0353 V	222	670						
SETDMA	0299 L	264	315	356					
SETDMAB	029D L	262	317	360					
SETSEC	0295 L	246	352						
SETTRK	0291 L	240	348						
SIGNONMESSAGE	0100 V	22	528						
SOURCE	0351 V	49	53	77	172	668			
SOURCEASCII	0190 V	51	559						
SOURCEMESSAGE	0135 V	35	540						
SPACEMESSAGE	0278 V	237	613						
SPT	03C3 V	112	123	125	252	269	301	695	
STACKEND	04D2 V	31	707						
START	0000 L	21							
TRACKBLOCK	0188 L	173	178	183	219				
TRACKSIZE	035C V	115	135	675					
TRK	0354 V	226	228	235	239	250	285	305	671
TYPEERRORMESSAGE	02C8 V	215	632						
VER	0014 N	4	538	538					
VERIFY	0263 L	181	313						
VERIFYBUFFER	04D2 V	314	321	710					
VERIFYGOOD	0285 L	326	328						
VERIFYMESSAGE	0266 V	180	609						
WRITE	02A5 L	176	368						
WRITINGMESSAGE	0254 V	175	605						

 COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

